ORIGINAL

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200302164-1

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Chi-Keung LUK et al.

Application No.: 10/029,699

Filing Date: 12/18/2001

Confirmation No.: 3236

Examiner: J. N. To

Group Art Unit: 2195

Title: SOFTWARE CONTROLLED PRE-EXECUTION IN A MULTITHREADED PROCESSOR

Mail Stop Appeal Brief-Patents
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF

Transmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on 03/27/2006.

The fee for filing this Appeal Brief is (37 CFR 1.17(c)) $500.00.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.136(a) apply.

☐ (a) Applicant petitions for an extension of time under 37 CFR 1.136 (fees: 37 CFR 1.17(a)-(d)) for the total number of months checked below:

☐ 1st Month $120	☐ 2nd Month $450	☐ 3rd Month $1020	☐ 4th Month $1590

☐ The extension fee has already been filed in this application.

☒ (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.

Please charge to Deposit Account 08-2025 the sum of $ 500. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

☐ I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Alexandria, VA 22313-1450
Date of Deposit:

OR

☒ I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (571)273-8300.
Date of facsimile: 05/01/2006
Typed Name: Colleen F. Brown
Signature: Colleen F. Brown

Number of pages: 23

Respectfully submitted,
Chi-Keung LUK et al.
By: Jonathan M. Harris
Attorney/Agent for Applicant(s)
Reg No. : 44,144
Date : 05/01/2006
Telephone : (713) 238-8000

Rev 10/05 (ApiBrief)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellants:	Chi-Keung LUK et al.	§	Confirmation No.:	3236
Serial No.:	10/029,699	§	Group Art Unit:	2195
Filed:	12/18/2001	§	Examiner:	J. N. To
For:	Software Controlled Pre-Execution In A Multithreaded Processor	§	Docket No.:	200302164-1

APPEAL BRIEF

Mail Stop Appeal Brief – Patents
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Date: May 1, 2006

Sir:

Appellants hereby submit this Appeal Brief in connection with the above-identified application. A Notice of Appeal was filed via facsimile on March 27, 2006.

05/02/2006 JBALINAN 00000044 082025 10029699
01 FC:1402 500.00 DA

Appl. No. 10/029,699
Appeal Brief dated May 1, 2006
Reply to final Office action of January 18, 2006

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The real party in interest is the Hewlett-Packard Development Company (HPDC), a Texas Limited Partnership, having its principal place of business in Houston, Texas. HPDC is a wholly owned affiliate of Hewlett-Packard Company (HPC). HPC merged with Compaq Computer Corporation (CCC) which owned Compaq Information Technologies Group, L.P. (CITG). The Assignment from the inventors to CITG was recorded on December 18, 2001, at Reel/Frame 012422/0760.

II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals or interferences.

III. STATUS OF THE CLAIMS

Originally filed claims:	1-36.
Claim cancellations:	None.
Added claims:	None.
Presently pending claims:	1-36.
Presently appealed claims:	1-36.

IV. STATUS OF THE AMENDMENTS

No claims were amended after the final Office Action dated January 18, 2006.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Appellants disclosure relates to generally to multi-threaded processors. Figure 3 is illustrative of Appellants' contribution and is explained in Appellants' disclosure as follows:

> Figure 3 shows the same portion of a program in two different forms 202 and 220. Each program form 202, 220 includes two program segments 204 and 206. Program 202 represents the program without the ability to spawn pre-execution threads. Program 220 represents program 202 modified to spawn pre-execution threads at desired points during program execution. As shown, the modifications include start and stop pre-execution thread instructions 222 and 226, as well as a label 224. In accordance with the preferred embodiment, the instruction to start a pre-execution thread is of the form "PreExecute_Start(label)" where "label" is a value that informs the program counter 106 where to begin executing the pre-execution thread. In the example of Figure 3, "label" is "LIST2" which is identified by numeral 224. Thus, at run-time, when PreExecute_Start(LIST2) is executed by the main thread, a pre-execution thread will be spawned to execute the program starting at the label LIST2. The pre-execution thread then continues executing code segment 206 from LIST2 until the PreExecute_Stop() instruction 226 is encountered.
>
> While the main thread is executing code segment 204, the pre-execution thread runs ahead of the main thread and executes code segment 206. Cache misses encountered during the execution of segment 206 by the pre-execution thread preferably are resolved before the main thread encounters the same memory references when it executes code segment 206. Broadly speaking, the pre-execution thread resolves one or more memory references and causes the requested data to be in data cache 146 before the main thread needs the data.

Appellants' disclosure page 9.

Appellants thus clearly explain that code segment 206 (which is a portion of a program as shown) executes in a pre-execution thread and in the main thread, but begins execution earlier in the pre-execution thread than in the main thread. Thus, the main thread executes an instruction 222 which spawns a pre-execution thread to begin executing a portion of the code that the main thread will execute in the future, but has not yet started executing. In at least some embodiments, such as the embodiment of Figure 3, the pre-execution thread

does not execute the entire program that is running in the main thread, but only select portions of the program that are specified by the main thread. Those portions that are selected for running in the pre-execution thread begin execution before their counterparts in the main thread begin execution. Appellants' contribution permits a software developer to modify software to run more efficiently in that the pre-execution thread will resolve such problems as branch mispredictions and cache misses thereby reducing such problems from occurring in the main thread.

In accordance with the invention of claim 1, a computer system comprises a processor 100 capable of executing multiple threads and a main system memory 92 coupled to the processor. As illustrated in Figure 3, the processor 100 processes a program 220 in a main thread that includes instructions (e.g., instruction 222) which cause the processor to spawn a pre-execution thread in which only a portion, but not all (portion 206), of the same program executes. The pre-execution thread runs concurrently with the main thread, but ahead of the main thread in program order. See also Disclosure page 6, para. [0018] and pages 8-10, paras. [0027] – [0032].

In accordance with the invention of claim 10, a processor 100 comprises a fetch unit 102, a program counter 106, an instruction cache 110 and a data cache 146. The fetch unit is capable of fetching instructions from a plurality of thread. The program counter is coupled to the fetch unit. The instruction cache is coupled to the fetch unit and the data cache is coupled to the instruction cache. The processor 100 processes a program in a first thread that includes instructions (e.g., instruction 222) which cause the processor to spawn a second thread in which only a portion, but not all, of the same program also executes. The second thread runs concurrently with the first thread, but ahead of the first thread in program order. See also Disclosure page 6, para. [0018] and pages 8-10, paras. [0027] – [0032].

In accordance with the invention of claim 19, a method of running a program in a processor comprises inserting pre-execution thread instructions (e.g., instruction 222) in the program, spawning a pre-execution thread when

designated by the inserted instructions, and running the pre-execution thread concurrently with a main thread. Both the pre-execution and the main threads include instructions from the same program. The pre-execution thread runs ahead of the main thread and contains only a portion of the instructions from the main thread. See also Disclosure page 6, para. [0018] and pages 8-10, paras. [0027] – [0032]. An example of this method is also illustrated in Figure 4.

In accordance with the invention of claim 31, a processor 100 comprises a fetch unit 102 capable of fetching instructions from a plurality of threads, a program counter 106 coupled to the fetch unit, an instruction cache 110 coupled to the fetch unit, and a data cache 146 coupled to the instruction cache. In a pre-execution thread, the processor pre-executes instructions from a main thread that are specified by the main thread. See also Disclosure page 6, para. [0018] and pages 8-10, paras. [0027] - 0032].

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Whether claims 1-30 comply with the written description requirement of 35 U.S.C. § 112, first paragraph.

Whether claims 1-36 are obvious (35 U.S.C. § 103) over a publication to Rotenberg in view of Georgiou (U.S. Pat. No. 6,032,245).

VII. ARGUMENT

A. The § 112, first paragraph, rejections of claims 1-30

Independent claims 1 and 10 recite the limitation that "only a portion, but not all, of the same program." Independent claim 19 includes a similar limitation. In the final Office Action, the Examiner rejected these claims alleging that the specification does not support this limitation and thus, these claims contain subject matter which was not described in a way as to reasonably convey to one skilled in the art that the inventors had possession of the claimed invention. Final Office Action pages 2-3.

As explained above with regard to the embodiment illustrated in Appellants' Figure 3, and associated text (supra), a main thread executes an instruction 222 which spawns a pre-execution thread to begin executing a segment 206 of program 220 beginning at label 224 and ending at stop instruction 226—a segment of code that the main thread will execute in the future, but has not yet started executing. Thus, the pre-execution thread does not execute the entire program 220 that is running in the main thread, but only select a portion (e.g., segment 206) of the program as specified by an instruction in the main thread. One of ordinary skill in the art would clearly understand that the subject matter of claims 1-30, which include the limitation "only a portion, but not all, of the same program" (claim 19 having a similar limitation) was in the possession of the inventors.

Based on the foregoing, Appellants respectfully submit that the § 112, first paragraph, rejections of claims 1-30 be reversed, and the claims set for issue.

B. The § 103 rejections of claims 1-36

1. Rotenberg

Rotenberg refers to a processor in which to instruction streams—an active (A) stream and a redundant (R) stream—are executed. See Figure 2 and associated text of Rotenberg. That is, the same program is executed twice in its entirety through the same processor, once in the A stream and again in the R stream. Rotenberg does not disclose that the A stream includes an instruction that spawns the R stream. The Abstract of Rotenberg explains that the "program

is duplicated and the two redundant programs simultaneously run on the processor."

2. Georgiou

Georgiou discloses a multi-processor system, not a multi-threaded processor. Figure 1 of Georgiou discloses three processors 16, 161, and 162 and a bus controller 110. The Examiner referred to a passage in column 3 of Georgiou which provides that "different threads, or portions, of a program execute on different processors 160-163 in parallel, each having its own program counter." Col. 3, lines 3-5.

3. Claims 1-30

Appellants select claim 1 as exemplary of the error of the Examiner's rejections of this claim group. Claim 1 requires that the:

> processor processes a program in a main thread that includes instructions which cause the processor to spawn a pre-execution thread in which only a portion, but not all, of the same program executes, said pre-execution thread runs concurrently with the main thread, but ahead of the main thread in program order.

Appellants respectfully submit that the Examiner's rejection is flawed for at least the following reason. Claim 1 requires that the program in the main thread include an instruction which causes a pre-execution thread to be spawned. No such instruction is disclosed as being included in Rotenberg's A stream. That is, the A stream of Rotenberg does not include an instruction that spawns the R stream. Georgiou also does not disclose a main thread including an instruction that causes a pre-execution thread to be spawned. Accordingly, Appellants respectfully submit that the rejections of the claims in this grouping be reversed, and the grouping set for issue.

4. Claims 31-36

Independent claim 31 specifies "wherein, in a pre-execution thread, said processor pre-executes instructions from a main thread that are specified by said main thread." Rotenberg does not teach or suggest that the A-stream specifies which instructions are to be pre-executed. Georgiou is also deficient with regard

to this claim limitation. Appellants thus respectfully submit that the rejections of the claims in this grouping be reversed, and the grouping set for issue.

5. Claims 2, 3, 11, 12, 20, 21,

Dependent claims 2, 3, 11, 12, 20, and 21 were erroneously rejected based on the reasons explained above regarding their independent claims. Claims 2, 3, 11, 12, 20, and 21 were also erroneously rejected for an additional reason. Appellants select claim 2 to illustrate this additional reason.

Claim 2 requires that the instructions included in the main thread include a start instruction and a stop instruction. The start instruction starts the pre-execution thread executing and the stop instruction which causes the pre-execution thread to stop. Neither Rotenberg nor Georgiou teaches such start and stop instructions, and certainly not as part of a main thread to spawn a pre-execution thread. Appellants thus respectfully submit that the rejections of the claims in this grouping be reversed, and the grouping set for issue.

C. Conclusion

For the reasons stated above, Appellants respectfully submit that the Examiner erred in rejecting all pending claims. It is believed that no extensions of time or fees are required, beyond those that may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required (including fees for net addition of claims) are hereby authorized to be charged to Hewlett-Packard Development Company's Deposit Account No. 08-2025.

Respectfully submitted,

Jonathan M. Harris
PTO Reg. No. 44,144
CONLEY ROSE, P.C.
(713) 238-8000 (Phone)
(713) 238-8008 (Fax)
ATTORNEY FOR APPELLANTS

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
Legal Dept., M/S 35
P.O. Box 272400
Fort Collins, CO 80527-2400

VIII. CLAIMS APPENDIX

1. (Previously presented) A computer system, comprising:

a processor capable of executing multiple threads; and

a main system memory coupled to said processor;

wherein said processor processes a program in a main thread that includes instructions which cause the processor to spawn a pre-execution thread in which only a portion, but not all, of the same program executes, said pre-execution thread runs concurrently with the main thread, but ahead of the main thread in program order.

2. (Original) The computer system of claim 1 wherein said instructions that cause the pre-execution thread to be spawned include a start instruction which causes a pre-execution thread to start and a stop instruction which causes the pre-execution thread to stop.

3. (Original) The computer system of claim 2 wherein said start instruction includes a value designating the location in the program where the pre-execution thread is to start running.

4. (Original) The computer system of claim 1 wherein the pre-execution thread encounters a cache miss condition for a memory reference but the main thread does not encounter a cache miss condition when that same memory reference is processed by the main thread.

5. (Original) The computer system of claim 1 wherein said processor determines whether sufficient hardware resources are available before spawning said pre-execution thread.

6. (Original) The computer system of claim 1 wherein said processor ignores exception conditions generated during the pre-execution thread.

7. (Original) The computer system of claim 1 wherein said processor does not permit a store instruction in the pre-execution thread to modify main system memory contents.

8. (Original) The computer system of claim 1 wherein said processor does not permit any store instruction in the pre-execution thread to modify main system memory contents.

9. (Original) The computer system of claim 1 further including a buffer into which pre-execution thread stores data is written to make such store data available to pre-execution thread load instructions.

10. (Previously presented) A processor, comprising:
a fetch unit capable of fetching instructions from a plurality of threads;
a program counter coupled to said fetch unit;
an instruction cache coupled to said fetch unit;
a data cache coupled to said instruction cache;
wherein said processor processes a program in a first thread that includes instructions which cause the processor to spawn a second thread in which only a portion, but not all, of the same program also executes, said second thread runs concurrently with the first thread, but ahead of the first thread in program order.

11. (Original) The processor of claim 10 wherein said instructions that cause a second thread to be spawned include a start instruction which causes the second thread to start and a stop instruction which causes the second thread to stop.

12. (Original) The processor of claim 11 wherein said start instruction includes a value designating the location in the program where the second thread is to start running.

13. (Original) The processor of claim 10 wherein the second thread encounters a cache miss condition for a memory reference but the first thread does not encounter a cache miss condition when that same memory reference is processed by the first thread.

14. (Original) The processor of claim 10 wherein said processor determines whether sufficient hardware resources are available before spawning said second thread.

15. (Original) The processor of claim 10 wherein said processor ignores exception conditions generated during the second thread.

16. (Original) The processor of claim 10 wherein said processor does not permit a store instruction in the second thread to modify data cache contents.

17. (Original) The processor of claim 10 wherein said processor does not permit any store instruction in the second thread to modify data cache contents.

18. (Original) The processor of claim 10 further including a buffer into which pre-execution thread store data is written to make such store data available to pre-execution thread load instructions.

19. (Previously presented) A method of running a program in a processor, comprising:

(a) inserting pre-execution thread instructions in the program;

(b) spawning a pre-execution thread when designated by the inserted instructions; and

(c) running said pre-execution thread concurrently with a main thread wherein both the pre-execution and the main threads include instructions from the same program, the pre-execution thread

running ahead of the main thread and containing only a portion of the instructions from the main thread.

20. (Original) The method of claim 19 further including stopping the pre-execution thread when designated by the inserted instructions in (a).

21. (Original) The method of claim 19 further including copying register contents associated with the main thread to registers used by the pre-execution thread.

22. (Original) The method of claim 19 further including determining whether sufficient hardware resources are available to spawn the pre-execution thread.

23. (Original) The method of claim 19 further including ignoring all exception conditions generated during the pre-execution thread.

24. (Original) The method of claim 19 further including not permitting a store instruction in the pre-execution thread to modify memory contents.

25. (Original) The method of claim 19 further including copying the contents of at least one register to memory to make such contents available to the pre-execution thread.

26. (Original) The method of claim 19 further including writing pre-execution store data into a buffer that can be accessed by a pre-execution load instruction.

27. (Original) The method of claim 19 further including not permitting any store instruction in the pre-execution thread to modify memory contents.

28. (Original) The method of claim 19 wherein (a) includes inserting a start instruction which causes the processor to start the pre-execution thread.

29. (Original) The method of claim 19 wherein the start instruction specifies a location in the program at which the pre-execution thread is to start running.

30. (Original) The method of claim 19 wherein (a) includes inserting a stop instruction which causes the processor stop the pre-execution thread.

31. (Previously presented) A processor, comprising:
a fetch unit capable of fetching instructions from a plurality of threads;
a program counter coupled to said fetch unit;
an instruction cache coupled to said fetch unit;
a data cache coupled to said instruction cache;
wherein, in a pre-execution thread, said processor pre-executes instructions from a main thread that are specified by said main thread.

32. (Previously presented) The processor of claim 31 wherein the pre-execution thread is caused to be spawned to pre-execute said instructions by an instruction in the main thread.

33. (Previously presented) The processor of claim 31 wherein the pre-execution thread spins on a variable that is set to a predetermined value by the main thread when there are instructions to pre-execute.

34. (Original) The processor of claim 31 wherein the processor ceases pre-executing instructions when a program counter is encountered that exceeds a range.

35. (Original) The processor of claim 31 wherein the processor ceases pre-executing instructions when the main thread catches up to the pre-executing instructions.

36. (Original) The processor of claim 31 wherein the processor ceases pre-executing instructions when the number of pre-executing instructions exceeds a limit.

IX. EVIDENCE APPENDIX

None.

X. RELATED PROCEEDINGS APPENDIX

None.